# COMPUTER VISION FOR DRIVER ASSISTANCE SYSTEMS

A Thesis
Presented to
The Academic Faculty

By

Sandhya Sridhar

In Partial Fulfillment
of the Requirements for the Degree
Master of Science in the
School of Electrical and Computer Engineering

Georgia Institute of Technology

May 2018

**COMPUTER VISION FOR DRIVER ASSISTANCE SYSTEMS**

Approved by:

Dr. David Taylor, Advisor
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Dr. Patricio Antonio Vela
School of Electrical and Computer Engineering
*Georgia Institute of Technology*

Dr. James Hays
College of Computing
*Georgia Institute of Technology*

Date Approved: April 25, 2018

Excellence is a continuous process and not an accident

*Dr. A. P. J. Adbul Kalam*

To my family who made this possible.

## ACKNOWLEDGEMENTS

Every project, big or small is successful largely due to the effort of a number of wonderful people who have always given their valuable advice or lent a helping hand. I sincerely appreciate the inspiration, support and guidance of all the people who have been instrumental in making this thesis a success.

Foremost, I am extremely grateful to my thesis advisor Dr. David Taylor, for his valuable advice and guidance throughout my thesis. I thank him for his continuous support and for keeping me on the right track, whilst allowing me the room to work in my own way. I also want to thank him for the confidence and trust bestowed upon me.

I profusely thank my thesis reading committee, Dr. Vela and Dr. Hays for taking time off their busy schedule and reading through my thesis. I also thank Dr. Hays for introducing me to Computer Vision through his lectures.

A big thanks to the EcoCAR 3 coordinators and sponsors, for giving me an exciting problem to solve and for their critical advice and support at the right time. I would also like to thank the entire Georgia Tech EcoCAR3 team for their help and support for this project.

A special thanks to my family, for their constant encouragement and support without which this thesis wouldn't have been possible.

## TABLE OF CONTENTS

## LIST OF FIGURES

## SUMMARY

The objective of this thesis is to illustrate the training, validation and evaluation of vehicle detection algorithms using computer vision and deep learning methods, and vehicle tracking in video sequences. The research work focuses on a traditional machine learning approach and a deep learning approach for detecting vehicles, and object tracking algorithms to improve the accuracy of object detection. Important excerpts of code for all the methods discussed are explained in the thesis and the complete code for the same is provided in the Appendix.

# CHAPTER 1
# INTRODUCTION AND BACKGROUND

## 1.1 Overview

Autonomous driving and driver assistance systems are becoming major areas of research in the automotive industry. With the recent success of Deep Learning, and the availability of better hardware and huge datasets, deploying deep learning to object detection is being widely researched, and many notable solutions have evolved over the last few years. Object detection involves localizing the object and finding its exact coordinates, in addition to determining its presence in the image. The chosen algorithm for object detection must also be feasible for real time detection for applications involving autonomous driving.

The classical object detection framework proposed by Viola and Jones [1] was one on the first object detection algorithms to run in real-time, and is still being used in point-and-shoot cameras for face detection. Another popular machine learning algorithm for object detection was proposed by Dalal-Triggs [2] around the same time. It uses Histogram of Oriented Gradient (HOG) features, as opposed to the Haar-like features proposed by Viola-Jones. This is much more accurate, but slower compared to the Viola-Jones algorithm.

With the recent success of deep learning, most of the present object detection algorithms used in the vehicles are powered by a neural networks. Faster RCNN [3] is a state-of-the-art framework for object detection, and is widely used along with convolutional neural networks for localizing objects in a frame. Other frameworks like YOLO (You Only Look Once) and SSD (Single Shot Detectors) [4] are also used for object detection in real-time scenarios based on the requirements. Object tracking algorithms are used along with object detection, to improve the accuracy of the detection. Various computer vision approaches like optical flow, feature tracking and motion vector estimation have been proposed for

tracking objects which have a steady motion. Kalman filter is also a popular choice for object tracking, as they predict the object's future state.

## 1.2 Thesis Organization

The research focuses on object detection strategies including a machine learning approach using a combination of Viola-Jones and Dalal-Triggs in the form of cascaded classifiers, and a deep learning based max-margin object detection method for detecting vehicles. Object detectors are trained, and their performance is tested on vehicle data recorded for the EcoCAR 3 competition in a test car. For vehicle tracking, the research covers a Kalman filter tracking and tracking objects based on correlation filters.

Chapters 2 and 3 highlight the approaches used for object detection. Chapter 2 explains in detail the machine learning approach to object detection. It provides details about using a cascade object detector to find the bounding boxes around receding vehicles.

Chapter 3 shows increased efficiency in vehicle detection using a deep learning approach. A trained deep neural network is used for predicting the bounding boxes around vehicles. The chapter provides an overview about the state of the art deep learning architectures used for object detection, and highlights the architecture of the network used, along with the results obtained.

Chapters 4 and 5 switch gears to explain object tracking methods. Chapter 4 introduces the reader to a motion estimation technique called optical flow, which can be used for tracking objects using a stationary camera. To aid real-time object tracking from a moving vehicle, Chapter 5 explains at length the tracking of multiple moving targets using a Kalman filter.

The thesis concludes with Chapter 6 which discusses the inferences from the two object detection and tracking methods explained in the previous chapters, and proposes future work. Finally, the Appendix contains the MATLAB code used for object tracking using Kalman filter, and the C++ code used for DNN based object detection.

# CHAPTER 2

# OBJECT DETECTION - MACHINE LEARNING

Object detection deals with finding instances of real-world objects, while handling variation with respect to the objects' shape, orientation and motion. Object detection algorithms typically take in a raw image as input and output the bounding box coordinates for the detected object. Object detection plays a significant role in programming autonomous vehicles, as it helps the vehicle perceive the environment just like a human driver would do. In this chapter, a classic machine learning approach using cascade object detectors is used for classifying images and detecting objects.

## 2.1 Cascade Object Detector

Object detectors can be trained to detect various objects which have distinct features and aspect ratio, using the cascade architecture proposed by Viola-Jones. The trained detectors are then used to detect objects by sliding a window over the image. Cascade detectors are a concatenation of many weak object classifiers that are placed sequentially, such that the output of a given classifier is sent as additional information to the next classifier. Each weak linear classifier forms a stage, and the final object detector combines these stages along with a sliding window detector.

For detecting the rear portion of cars, a 13-stage cascade object detector with the Histogram of Oriented Gradient features is trained. Many 'positive images' containing the object to be detected and 'negative images' which do not contain the object are fed to these cascaded classifiers as inputs. A sliding window search is performed over the input images, and each window is classified as positive or negative, based on the presence of the car's rear. Figure 2.1 shows an overview of the machine learning approach that is followed to detect vehicles in the image.

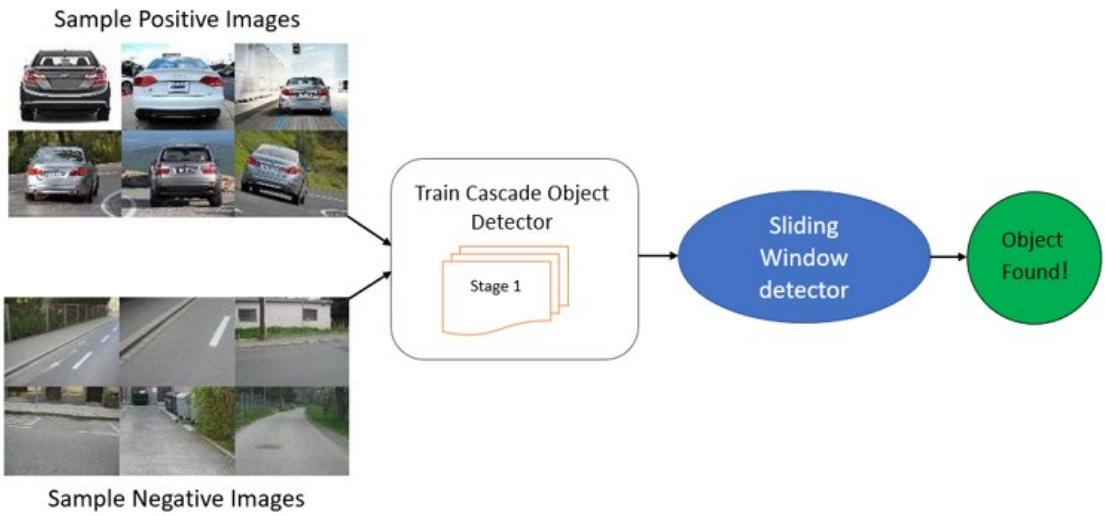



Figure 2.1: Overview of Machine Learning Approach

## 2.2 Feature - Histogram of Oriented Gradients

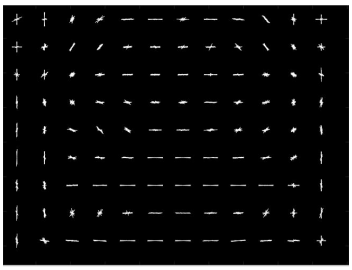
A feature descriptor or feature vector is a representation of an image or an image patch that simplifies the image by extracting useful information and encoding them in a way that they are unique for each image. There are several feature descriptors like SIFT, SURF Haar-like features and HOG (Histogram of Oriented Gradients). HOG features are chosen here, as they capture the shape of the object better than other features. Since the features chosen for this approach are sensitive to the aspect ratio of the object to be detected, a separate detector must be trained for each orientation of object.

In the HOG feature descriptor, the distribution of directions of oriented gradients are used as features to represent images. Around the edges and corners of an image, the magnitude of intensity changes is large. These intensity changes reveal a lot about the shape and orientation of the objects, and hence are extracted as useful gradient features.

To compute the HOG features, the image is broken into smaller windows, and in each window, every pixel is compared with its neighboring pixels, to compare the variation in brightness across the pixels. The gradient arrow is drawn along the direction in which the pixel intensity increases. The gradient direction that is the maximum in that window is

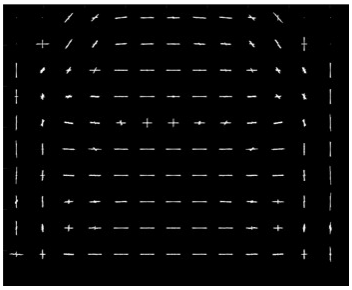
then assigned as the window's gradient direction. This process is repeated for the entire image, until all the windows are replaced by gradient arrows, which show the variation in pixel intensities from bright to dark. In contrast to analyzing the pixel intensities directly, computing the gradients is better as it is brightness invariant, and so Histogram of Oriented Gradients feature is chosen for vehicle detection. Figure 2.2b and 2.2d show the HOG pattern obtained for sample images shown in Figure 2.2a and 2.2c.

(a) Sample image of car's rear



(b) Visualization of HOG for the image


(c) Sample image of car's rear



(d) Visualization of HOG for the image

Figure 2.2: Histogram of Oriented Gradients Feature Extraction

With the obtained HOG pattern, the test image is compared to check for similar HOG patterns, to detect the rear of vehicles. For training an efficient cascade object detector, several parameters must be chosen optimally. MathWorks' and OpenCV's documentation [5] [6] on training a cascade object detector provide sufficient information on the working mechanism and training parameters. The parameters chosen for training are:

- **Number of stages:** The number of stages used depends upon the number of positive and negative samples used for training. The training terminates if the number of positive or negative samples is not sufficient to fit the next stage.
- **False Alarm Rate:** Lowering this would lead to fewer false detections.

- **True Positive Rate:** Minimum rate for each stage is chosen close to 1.
- **Feature:** For cascade object detectors, the supported features are Haar, HoG and LBP (Local Binary Pattern). HoG features are chosen here for vehice detection due to their better performance in capturing the orientation of the object.

Table 2.1 shows the values chosen for the above parameters in training the cascade object detector.

Table 2.1: Training parameters for cascade object detector

| Parameter | Value chosen |
|---|---|
| Number of positive samples | 1200 |
| Number of negative samples | 2100 |
| Number of stages | 13 |
| False Alarm Rate | 0.2 |
| True Positive Rate | 0.995 |
| Feature | HOG |

### 2.3 Snippet - Training a Cascade Object Detector

The Matlab code snippet below shows how a cascade object detector can be trained, to detect the rear of vehicles.

```
%% Load positive images and bounding boxes of bikes
load Car
%% Step 2: Specify folder with negative images
negativeFolder = [pwd '\noncars'];
%% Step 3: Train the detector
NumStages = 13;
FAR = 0.2;
TP = 0.995;
trainCascadeObjectDetector('CarDetector.xml', Car,...
```

```
10      negativeFolder, 'NumCascadeStages', NumStages,...
11      'FalseAlarmRate', FAR,'TruePositiveRate', TP);
```

### 2.4 Input Data - Camera Setup

A dashboard-mounted stereo camera assembly, comprising of 2 USB webcams is used to capture stereo images. The USB cameras used are Logitech C920 HD Pro Webcams, which come with a photo quality of 15MP, and a video quality of 1920 x 1080. The frame rate of the cameras is 30fps. The cameras are suitable for tripod mounting and are mounted on a Vanguard Multi-Mount Horizontal Bar. The relative distance between the two cameras is set to 12cm and can be adjusted as per the requirement up to 25cm. A stereo camera setup is used here, since the camera data is also used to predict the distance to the detected objects, as a separate part of the project outside the scope of this thesis. The images from the camera on the left are used for object detection and tracking, and the ground truth database is developed on those images.

### 2.5 Ground Truth

Visual observation to evaluate the object detector's performance becomes too time consuming in a real time scenario for videos due to the substantial number of frames. The detector must be tested for performance with variations in thresholds, morphological operations and changes in the number of stages, to find the optimal parameters for the task and to obtain the best possible object detector from the training dataset. To make the evaluation of the detector's performance easier in such scenarios, a ground truth database is developed and used for comparison with changes in the object detection algorithm.

Ground Truth refers to the user-defined labels and bounding boxes for known objects. Ground Truth database can be obtained in three ways as follows:

- Manual labeling

- Using a database
- Automated ground truth generation

For specific tasks like the EcoCAR 3 evaluation, manual labeling can be utilized since the objects to be labeled are unique, and the ground truth data can be developed from videos provided for the competition or the videos captured from our camera setup. For academic research, computer vision databases that are uploaded by the researchers can be used, to avoid spending excessive research time in labeling. Ground truth data can also be generated automatically by using a robust object detector. Figure 2.3 shows a screenshot of MATLAB's Ground Truth Labeler App used for manually developing the ground truth database.


Figure 2.3: Ground Truth Database Development

## 2.6 Object Detection Metrics

By using the developed ground truth data, the labeled bounding boxes and the object detector can be compared to evaluate the performance of the object detector. The Intersection-over-Union (IoU) in the area of the bounding boxes generated by the object detector and ground truth is compared to evaluate the object detector. An IoU value of 0.7 is treated as the desired output. The evaluation metrics used for evaluating the detector are:

- **True Positive Rate:** True Positive Rate (TPR), also called as Sensitivity corresponds to the number of positive objects correctly identified over the number of positive

objects labeled. This value must be high for a good object detector. TPR can be mathematically written as,

$$TPR = TP / (TP + FN)$$

where TP and FN correspond to True Positive and False Negative objects. This term is also referred to as Recall.

- **False Positive Rate:** False Positive Rate (FPR) is defined as the number of positive objects incorrectly identified over the total number of objects found. It can be written mathematically as,

$$FPR = FP / (FP + TP)$$

where FP and TP correspond to the False Positive and True Positive objects found. For a good object detector, the FPR must be as small as possible. The precision of an object detector can be calculated from the FPR as,

$$Precision = 1 - FPR$$

## 2.7 Results - Cascade Object Detector

The results obtained by running the trained cascade object detector on a sample video recorded using the camera setup mentioned above are shown below, in Figure 2.4. The calculated recall of the detector is 52%. The detector fails to detect the object in some intermediate frames in the sequence, and this can be improved by object tracking methods. Even objects that appear small in frames are detected in this approach.

The problem with this detector is its precision. The calculated precision of this detector on the sample video is 28%. This is because there are instances like the ones shown in the results, where the detector bounds the same area more than once. This can be improved by

changing the parameters of the non-maximum suppression algorithm to display only one bounding box per object.

Desired Results | Undesired Results

Figure 2.4: Results - Cascade Object Detector

The performance of this object detector can be improved by choosing the training dataset carefully and tuning the training parameters. However, to achieve state-of-the-art results in object detection, an object detector using the concepts of deep learning is trained, and the details are explained in Chapter 3.

# CHAPTER 3

# OBJECT DETECTION - DEEP LEARNING

Deep Learning and convolutional neural networks have become extremely popular in the recent times, due to the availability of huge datasets and better hardware processing capabilities. Deep learning object detector models are the state of the art in computer vision tasks and are used to assist the autonomous vehicle perceive the environment just like a human driver. The perception algorithm fuses data from various sensors like the Camera, RADAR and LIDAR, and then helps the vehicle plan its motion and drive autonomously.

## 3.1 Object Detection Challenges

Object detection combines the tasks of classification and localization, and outputs the object along with its pixel coordinates in the input image. This poses a few challenges as,

- **Varying number of objects:** The number of objects detected in an input image is not predefined, and due to this the output of the detector cannot be a vector of a fixed size. As discussed in Chapter 3, traditional machine learning approaches use a sliding window detector to find all the objects in the image.

- **Size of the objects:** In image classification tasks, the biggest object in the image is classified into one of the known categories of objects. As opposed to this, objects of varying sizes must be detected in a detection task. In traditional machine learning approaches, sliding windows of varying sizes are used to detect objects in various scales.

- **Architecture:** An object detection task combines object classification and localization. The architecture used for detecting objects must thus include the requirements of both these tasks into consideration and a combined model must be created.

## 3.2 Literature Survey - Deep Learning Approaches

A lot of research and development has been happening in using deep learning for object detection during the past few years. One of the earliest methods is the OverFeat[7], published in 2013. It shows how a multi-scale sliding window can be used efficiently in a convolutional neural network (ConvNet) to detect objects. Soon after OverFeat's publication, Ross Girshick published the R-CNN[8] (Region based Convolutional Neural Networks) and this approach had nearly a 50% improvement to OverFeat. In this approach, possible objects and regions are extracted and then the regions are classified. R-CNN evolved into Fast R-CNN in which a CNN is applied to the entire image instead of extracting the object and region proposals separately. To make process significantly faster, the Faster R-CNN[3] was then introduced. Instead of the selective search that was used in the previous two R-CNN approaches for region proposals, the Faster R-CNN uses a 'Region Proposal Network' which makes the entire model completely trainable end-to-end. Another object detection approach, You Only Look Once (YOLO)[9] uses a single CNN to predict the bounding boxes and class probabilities from the input image. This method has a great inference time and allows object detection to be performed in real-time. SSD[4] (Single Shot Multi-box detection) is a method that improved upon YOLO and used multiple sized convolutional feature maps while R-FCN (Region based Fully Convolutional Network) improved upon Faster RCNN by using only convolutional layers.

## 3.3 Background - Deep Learning

One of the main ideas behind deep learning is automatic feature extraction. Deep learning aims to automatically learn the features from noisy data - this is a major challenge for traditional machine learning and the performance of a machine learning algorithm drops significantly with noisy data. A deep network is usually made up of an input layer, many hidden layers that learn the high-level features using some activation functions, followed

by an output layer. While training, random values are assigned as the network's weights, and the network learns the optimal weights using an algorithm called the backpropagation. In the simplest sense, backpropagation algorithm propagates the errors from output back to the input and calculates the partial derivatives with respect to each weight to compute steps in the forward direction. As more and more hidden layers are added to the network, the weight update process becomes harder with decreasing input signal strength. Deep learning helps alleviate this issue by using various algorithms for feature extraction.

### 3.4 DNN Architecture for Vehicle Detection

A Deep Neural Network (DNN) along with Max-Margin Object Detection (MMOD) is used for detecting vehicles, as implemented in the dlib library. The overall process used for vehicle detection using this approach is shown in Figure 3.1. An RGB image is taken in as input, and it is downsampled to create a Gaussian pyramid of images. These images are arranged in a tile and passed in to the DNN, which learns to detect the vehicles in the images by computing the MMOD losses. The output bounding boxes from the DNN are placed on the collapsed pyramid and non-maximum suppression is performed to remove multiple boxes that correspond to the same object, and the boxes with the highest confidence are displayed on the input image.

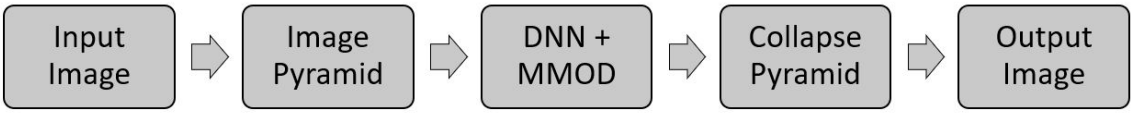



Figure 3.1: Vehicle Detection using Deep Learning

The following subdivisions explain the process involved in vehicle detection using the above architecture.

### 3.4.1 Input Image

Each frame of the recorded input video is passed in as input to the DNN based vehicle detector. The 3-channel input image is of size 1280 x 720 and is shown below in Figure 3.2.

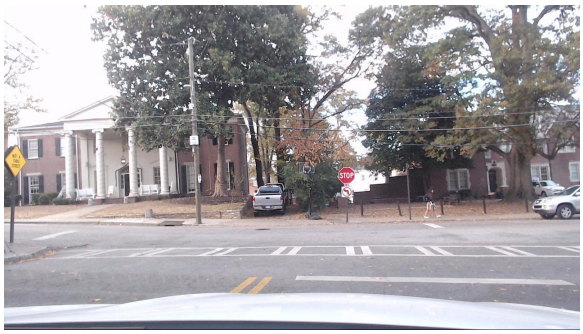


Figure 3.2: Input Image

### 3.4.2 Creating an Image Pyramid

In computer vision, two types are image pyramids are popularly used based on the kernels - Gaussian and Laplacian pyramids. Here, a Gaussian pyramid is created from the input image using a Gaussian blur function. The input image is downscaled by a factor of 5/6, and a tiled pattern is created from the input images. The tiled image created by arranging all the downscaled images together is approximately 3.7x times larger in size than the original input image, and the 3 channels of this tiled image are passed in as input to the network. Figure 3.3 shows the tiled pyramid formed with the input image.

Figure 3.3: Tiled Pyramid

### 3.4.3 Convolutional Neural Network

A 7-layer convolutional neural network is used for the detection of vehicles. Three convolutional layers with a stride of 2 are used to downsample the image pyramid, it downsamples the input to a factor of 8. Four convolutional layers with a stride 1, are used to extract useful features from the input. Stride is the number of pixels by which the filter matrix slides over the input matrix. ReLU (Rectified Linear Unit) is used as the activation function, to introduce non-linearity in the output of a neuron in the network. This is done since the real-world data is mostly non-linear, and the network should learn the non-linear representations to perform well on real data.

Figure 3.4 shows the architecture of the DNN used in the library, consisting of Convolution layers and ReLU activation functions. The network consists of downsampling and feature extraction layers and processes the three channels of the RGB input image and outputs bounding boxes. A sliding window search is used for localization, to find the co-ordinates of the detected objects.

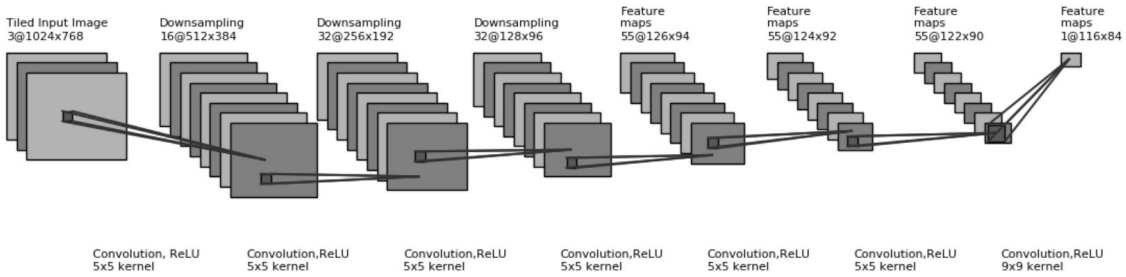



Figure 3.4: DNN - Architecture

Figure 3.5 shows the intermediate output from the neural network. This image is obtained by overlaying the network's output in the form of a color jet on the tiled pyramid.

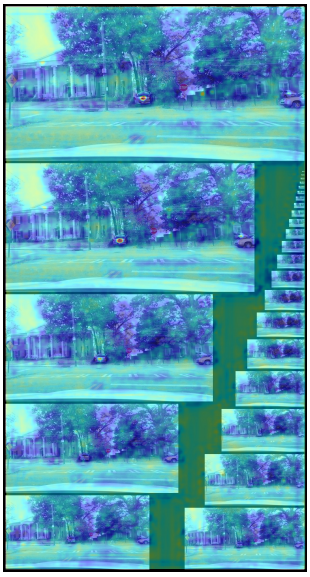


Figure 3.5: DNN Output

### 3.4.4 Collapsing the Image Pyramid

The bright red blobs found on the image pyramid are the places where the network outputs a positive label for a vehicle, and blue denotes the absence of the target object. Although thresholding the tiled pyramid is sufficient to get the detections, the pyramid is collapsed, and the detections are marked on a single image. The processed image shown in Figure 3.6 is also upsampled 8 times, to compensate the effect of the initial 3 layers of the pyramid.

Figure 3.6: Collapsed Pyramid

### 3.4.5 Output Image

The final output image is obtained by drawing bounding boxes over the red blobs, keeping them as the center of the detected area. The output of the vehicle detection task on a single input image is shown in Figure 3.7.

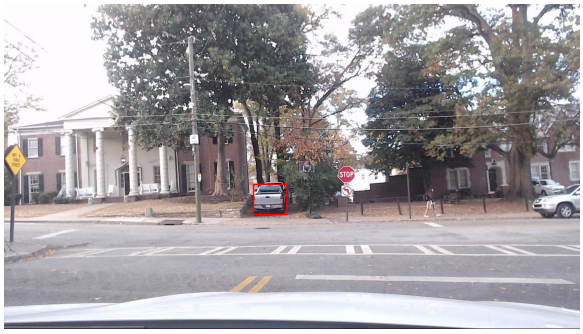


Figure 3.7: Output Image

The same process is extended for detecting vehicles in a video, by processing each frame in the input video.

## 3.5 Results - DNN Vehicle Detection

The DNN based detector explained above is run on the same sample video as the cascade object detector, and some of the results are shown below in Figure 3.8. The DNN based detector performs better than the former approach and has an observed recall and precision of 80% and 95% respectively. The detector fails to identify vehicles that have an orientation

over 60 degrees or vehicles smaller than 70 x 30 pixels. On the other hand, the detector finds even partially occluded vehicles, which is a crucial aspect of object detection systems.

Desired Results | Undesired Results

Figure 3.8: Results - DNN based Vehicle Detection

### 3.6 Comparison and Inference

Figure 3.9 highlights the significant improvements in object detection that are observed when using the DNN based detector instead of the cascade object detector. The cascade object detector fails to identify the vehicle in the first set of images and doesn't detect the occluded vehicles in the second set, while these cases are detected by the DNN based detector. In the third set of images, the cascade object detector finds the side portion of a car along with a false positive, even though the aim is to detect the rear portion of vehicles. The DNN detector on the other hand, detects the rear portion of the vehicle with a significant orientation. The cascade object detector outperforms the DNN based detector in the 4th pair of images. Although it has a false detection, the detector identifies the vehicle even from a distance. Since the minimum size of the object that the DNN based detector recognizes is 30 x 70, it fails to detect the vehicle in this frame. Since the cascade object detector is


Figure 3.9: Comparison of Results - Cascade Object Detector and DNN Detector

trained based on the shape of the object, it wrongly detects the front portion of the vehicle as its rear, as seen in the last pair of images.

# CHAPTER 4
# OBJECT TRACKING - OPTICAL FLOW

Object tracking is the process of estimating and detecting the motion of moving objects through consecutive frames in a video. Object tracking finds significant applications in many fields including automotive safety and autonomous driving, biomedical applications, surveillance and augmented reality. The goal of object tracking is to estimate the next position of the detected image, given its current position in pixel coordinates. In this chapter a method of object tracking based on the apparent movement of the pixels, known as Optical Flow is discussed.

## 4.1 Optical Flow

The apparent motion of pixels can be used to infer a lot of details about a set of sequential images, which makes optical flow a popular application for finding the motion of objects and for tracking in video processing. It has been an active area of research in the field of computer vision since it was first introduced by Berthold Horn and Brian Schunck in 1981[10].

Optical flow field is defined as the velocity vector field of apparent motion of brightness patterns in a sequence of images. When estimating the motion of an object, the motion between two subsequent frames is assumed to be small. This assumption helps in generating motion vectors, to calculate the strength and direction of the motion.

One of the popular methods in optical flow is Horn-Schunck. In this method, it is assumed that the motion of the brightness patterns is the result of relative motion, large enough to register a change in the spatial distribution of intensities on the images. Thus, relative motion between an object and a camera can give rise to optical flow. Since the images are separated by a small time step in a video sequence, the flow vector is suffi-

ciently smooth and has small displacements. Since optical flow estimates the motion of objects, it is also used in various applications like motion-based segmentation, recognition of gestures, surveillance and video compression.

## 4.2 Mathematical Formulation

This section gives the reader a background about the computation that occurs in Matlab's *OpticalFlowHS*[11] function, for calculating the Horn-Schunck optical flow vectors in an image.

### 4.2.1 Optical Flow Constraints

Because optical flow is a paradigm based upon visual perception, Horn and Schunck made several assumptions which match the human vision system. One of the main assumptions is the brightness constancy.

**Brightness Constancy**

A given grayscale image sequence can be represented as $I(x, y, t)$ where $(x, y)$ denote the image domain, and $t$ is the time parameter. Under this constraint, the changes in the image brightness is assumed to be 0, so that any movement observed will be due to the movement of the pixels in the image. Mathematically, the brightness constancy can be expressed as,

$$I(x(t), y(t), t) = \text{Constant}$$

So, differentiating with respect to time will be 0. This can be mathematically stated as,

$$\frac{\mathrm{d}I}{\mathrm{d}t} = 0$$

Differentating $I$ with respect to each component, this becomes,

$$\frac{\partial I}{\partial x}\frac{\mathrm{d}x}{\mathrm{d}t} + \frac{\partial I}{\partial y}\frac{\mathrm{d}y}{\mathrm{d}t} + \frac{\partial I}{\partial t} = 0$$

Here, the components of velocity are given by

$$\frac{\mathrm{d}x}{\mathrm{d}t} = u \text{ and } \frac{\mathrm{d}y}{\mathrm{d}t} = v$$

Using the above two equations, the brightness constancy term can be restated as,

$$I_x u + I_y v + I_t = 0$$

Since we are interested in the total change in these terms, the energy functional can be stated as,

$$E_b = \iint (I_x u + I_y v + I_t)^2 dxdy$$

where $E_b$ is the energy associated with the brightness constraint. This term is known as the fidelity term. This is the energy equation associated with grayscale images. As there are 2 unknowns and only 1 equation, this becomes an under-determined problem. Since RGB images are considered for the computation, this issue does not arise, and it ends up being an over-determined problem since there are 3 equations associated with each channel of R, G and B, while there are only 2 unknowns.

**Smoothness Constraint**

To make the problem well-posed for grayscale images, Horn and Schunck introduced a new constraint. The smoothness of the flow is penalized in this constraint, to make the flow smooth. This constraint states that the image intensity velocity field must vary smoothly everywhere except at object boundaries, where there can be discontinuities. The energy functional for the smoothness constraint is given by

$$E_s = \alpha \iint (|\nabla u|^2 + |\nabla v|^2) dxdy$$

where $E_s$ is the energy associated with the smoothness constraint and $\alpha$ is the regularization term that is used to penalize the smoothness constraint. With the addition of a new

constraint, the problem becomes well posed even for grayscale images.

### 4.2.2 Energy Functional

The total energy to be minimized is the sum of the energy associated with the brightness and smoothness constraint. Since RGB images are considered for optical flow computation, the energy functional for the 3 channels can be defined as,

$$E(u,v) = \sum_{\text{R,G,B}} \iint ((I_x u + I_y v + I_t)^2 + \alpha(|\nabla u|^2 + |\nabla v|^2))dxdy$$

The other assumptions taken for grayscale images hold for RGB images as well. This type of minimization problem belongs to the Calculus of Variations, and the energy functional is given by the solution to a pair of Euler-Lagrange equations, as computed below.

**Euler-Lagrange Equations**

The energy functional can be thought of as $L(u, v, u_x, v_x, u_y, v_y, x, y)$, in terms of the Lagrangian. The Lagrangian can be expanded and written as,

$$L = {I_x}^2 u^2 + {I_y}^2 v^2 + {I_t}^2 + 2(I_x I_y uv + I_x I_t u + I_y I_t v)$$

$$+\alpha(u_x^2 + u_y^2 + v_x^2 + v_y^2)$$

The Euler-Lagrange equation for this problem takes the form,

$$\frac{\partial L}{\partial u} - \frac{\mathrm{d}}{\mathrm{d}x}\frac{\partial L}{\partial u_x} - \frac{\mathrm{d}}{\mathrm{d}y}\frac{\partial L}{\partial u_y} = 0$$

$$\frac{\partial L}{\partial v} - \frac{\mathrm{d}}{\mathrm{d}x}\frac{\partial L}{\partial v_x} - \frac{\mathrm{d}}{\mathrm{d}y}\frac{\partial L}{\partial v_y} = 0$$

Derivatives of $L$ in $u$, $v$, $u'$ and $v'$ are given as follows:

$$\frac{\partial L}{\partial u} = 2(I_x^2 u + I_x I_y v + I_x I_t)$$

$$\frac{\partial L}{\partial v} = 2(I_y^2 v + I_x I_y u + I_y I_t)$$

$$\frac{\partial L}{\partial u_x} = 2\alpha u_x$$

$$\frac{\partial L}{\partial u_y} = 2\alpha u_y$$

$$\frac{\partial L}{\partial v_x} = 2\alpha v_x$$

$$\frac{\partial L}{\partial v_y} = 2\alpha v_y$$

By substituting the above value of the derivatives, the resulting Euler-Lagrange equations are,

$$I_x^2 u + I_x I_y v + I_x I_t - \alpha u_{xx} - \alpha u_{yy} = 0$$

$$I_y^2 v + I_x I_y u + I_y I_t - \alpha v_{xx} - \alpha v_{yy} = 0$$

This can be written more compactly as,

$$(I_x u + I_y v + I_t) I_x - \alpha \nabla^2 u = 0$$

$$(I_x u + I_y v + I_t) I_y - \alpha \nabla^2 v = 0$$

The value of $u$ and $v$ can be obtained from the above equations by introducing a new variable $\tau$.

$$u_\tau = -\nabla_u E = \Delta u - I_x^2 u - I_x I_y v - I_x I_t$$

$$v_\tau = -\nabla_v E = \Delta v - I_y^2 v - I_x I_y u - I_y I_t$$

where $\Delta u$ and $\Delta v$ are the Laplacian operators on the $u$ and $v$ components of the flow field. In Matlab, the Horn-Schunck optical flow function can be used by calling the function *OpticalFlowHS*, which accepts the expected smoothness of the optical flow, maximum number of iterations and the minimum absolute velocity difference to stop iterative computation as

inputs and computes the optical flow vector using the equations above.

By thresholding the motion vectors, the model creates binary feature image containing blobs of moving objects. Median filtering is used to remove scattered noise. Close operation is performed to remove small holes in blobs. The model locates the cars in each binary feature image using the Blob Analysis block. Then it uses the Draw Shapes block to draw a green rectangle around the cars that pass beneath the white line. The counter in the upper left corner of the Results window tracks the number of cars in the region of interest.

### 4.3 Snippet - Estimating the Optical Flow Vectors

The Matlab snippet below shows how the Horn-Schunck optical flow vectors can be calculated for a given video frame.

```
1   vidReader = VideoReader('visiontraffic.avi', '
        CurrentTime', 11);
2   opticFlow = opticalFlowHS;
3   while hasFrame(vidReader)
4     frameRGB = readFrame(vidReader);
5     frameGray = rgb2gray(frameRGB);
6    % Compute optical flow
7     flow = estimateFlow(opticFlow, frameGray);
8     % Display video frame with flow vectors
9     imshow(frameRGB)
10    hold on
11    plot(flow, 'DecimationFactor', [5 5], 'ScaleFactor',
          60)
12    drawnow
13    hold off
14  end
```

## 4.4 Results - Optical Flow based Vehicle Tracking

Optical flow vectors are calculated using the above snippet and are plotted on the input image sequence shown below in Figure 4.1. The flow vectors are computed on a grayscale image. After defining the parameters of the optical flow method, Matlab's *estimateFlow* function can be used to compute the flow field.

Cascade Object Detector DNN based Detector

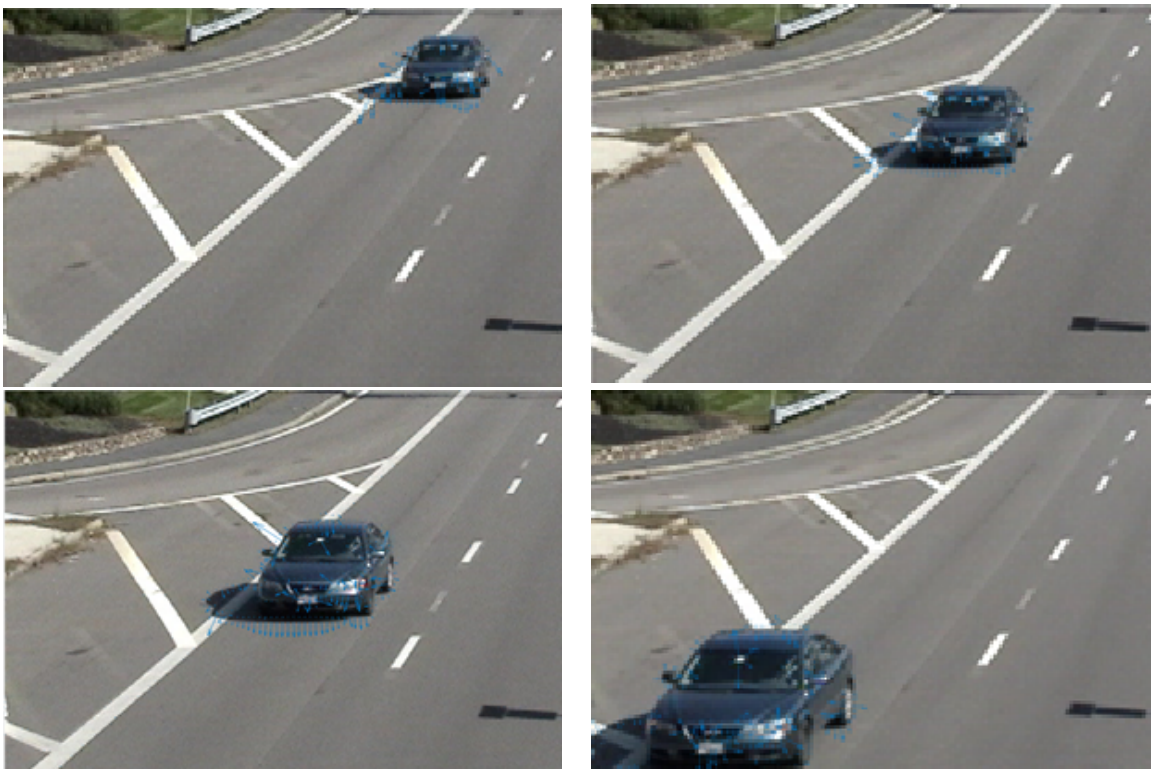
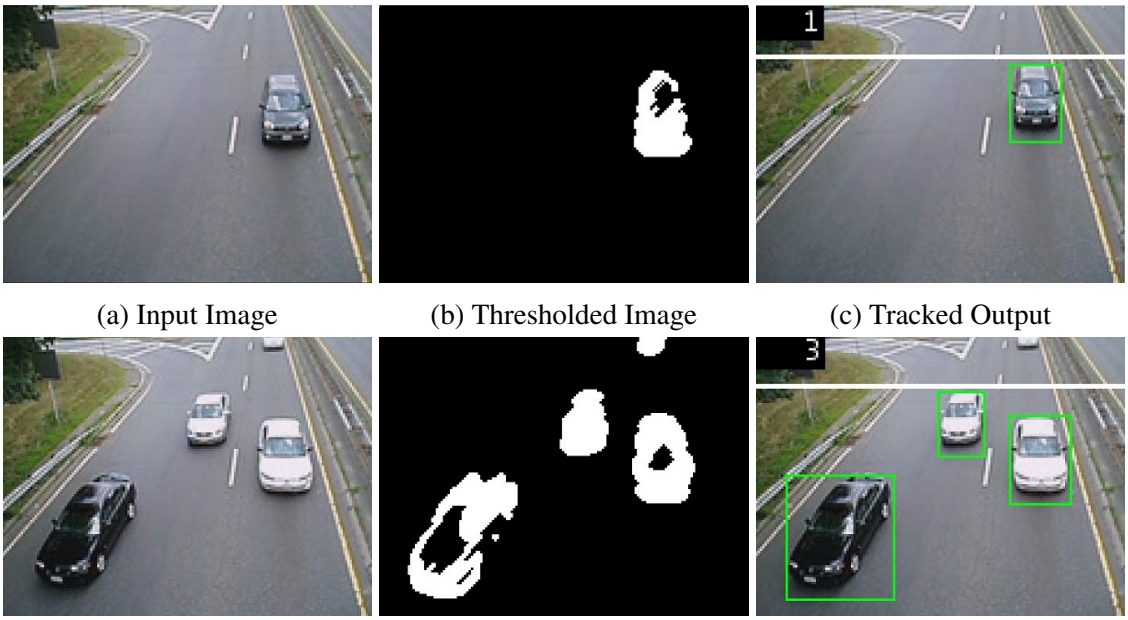


Figure 4.1: Visualization of Optical Flow Vectors

Optical flow vectors are calculated by thresholding the output vectors on a video recorded from a stationary camera. Using the motion vectors, blobs are created around the moving objects in each frame. The biggest blobs are filtered out as vehicles, and a bounding box is drawn around these blobs, along with a count of the number of vehicles tracked in each frame. The results of the optical flow object tracking method are shown in Figure 4.2.




(a) Input Image (b) Thresholded Image (c) Tracked Output

(d) Input Image (e) Thresholded Image (f) Tracked Image

Figure 4.2: Results of Optical Flow Tracking

## 4.5 Shortcomings of Optical Flow

Although optical flow provides a good estimate of the motion of objects, it has the following shortcomings with regards to object tracking:

- **Occlusion**

  Since optical flow tracks only the apparent movement of pixels between frames, the object is not tracked if it is occluded behind another object. It depends only on the motion of pixels, and hence does not detect and track an occluded object.

- **Movement of Camera and Object**

  This method works well when either the camera or the object is in motion, but not when there is a relative movement in both the camera and the object. For a real-time vehicle tracking system, this method cannot be used to effectively track other objects from a moving vehicle since the motion of pixels in the entire frame will be large.

- **Large Movements**

  Large movements may not be detected correctly because they produce great displacements instead of smooth motion. To detect large movements, either the frame rate must be increased, or the image size must be decreased.

- **Illumination**

  Changes in illumination can greatly affect the flow vectors and will lead to the creation of flow vectors at postions without motion. Since the method assumes brightness constancy over the image sequence, objects cannot be tracked when there is an illumination change in the images.

## 4.6 Other Optical Flow Models

Other than Horn-Schunck method of optical flow, there are various other methods that estimate the motion of pixels. Some of the most commonly used optical flow methods are,

- **Lucas-Kanade**

  In contrast to Horn-Schunck optical flow method, Lucas-Kanade method assumes local smoothness constancy in image sections. It is more robust to noise compared to Horn-Schunck, especially when using the derivative Gaussian filter. To obtain reliable flow vectors, this method takes more iterations compared to the Horn-Schunck and is comparatively slower to converge.

- **Farneback**

  This method also assumes a global smoothness constancy like the Horn-Schunck. With lesser number of iterations and a faster run-time compared to Horn-Schunck, the Farneback method of optical flow provides reliable flow vectors when tracking larger displacements. It is also more robust to noise compared to Horn-Schunck optical flow method.

# CHAPTER 5
# OBJECT TRACKING - KALMAN FILTER

In this chapter, object tracking based on the Kalman filter is discussed. This method can be used for effectively tracking multiple objects, and to predict their motion to track the object in the case of occlusion. This method is used in conjunction with object detection to refine the detections and improve detection efficiency by keeping track of the objects between frames.

## 5.1 Kalman Tracking - Introduction

In the simplest form, a Kalman filter takes in the current state of the system along with a measure of the uncertainty in the current measurement and predicts the next state of the system. The predicted state is then corrected (or updated) by comparing the predicted state and the next input. In object tracking, the position of a moving object at the next instant has to be predicted from the current position of the object. It estimates the internal state of the process (velocity), given a noisy observation (the extracted position of the object in pixel coordinates).

## 5.2 Mathematical Formulation

For vehicle tracking, a constant velocity motion model is chosen. The first derivative of position (velocity) is fairly constant, and the second derivative (acceleration) is almost zero. The motion of the object doesn't typically have a constant velocity or a constant acceleration. While programming the Kalman filter, white noise is added to it to account for the real-world situation. In 1D, the dynamics of an object moving with constant velocity is

given by

$$x_t = x_{t-1} + T\dot{x}_{t-1}$$

$$\dot{x}_t = \dot{x}_{t-1} + T\ddot{x}_{t-1}$$

where $x_t$, $\dot{x}_t$ and $\ddot{x}_t$ denote the position, velocity and acceleration of the moving object respectively and $T$ denotes the time span between two frames. Extending the dynamics of the system to 2D, the state variables $x_t$, $y_t$, $\dot{x}_t$ and $\dot{y}_t$ satisfy

$$x_t = x_{t-1} + T\dot{x}_{t-1}$$

$$y_t = y_{t-1} + T\dot{y}_{t-1}$$

$$\dot{x}_t = \dot{x}_{t-1} + T\ddot{x}_{t-1}$$

$$\dot{y}_t = \dot{y}_{t-1} + T\ddot{y}_{t-1}$$

which can be written in the form

$$\begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & T & 0 \\ 0 & 1 & 0 & T \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_{t-1} \\ y_{t-1} \\ \dot{x}_{t-1} \\ \dot{y}_{t-1} \end{bmatrix} + W_{t-1}$$

where $W$ is the motion noise. The measurement matrix for observing the position becomes

$$\begin{bmatrix} x_t \\ y_t \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_t \\ y_t \\ \dot{x}_t \\ \dot{y}_t \end{bmatrix} + V_t$$

where $V$ is the measurement noise. While programming a Kalman filter in Matlab[12], the function *configureKalmanFilter* can be used to set the initial parameters to track a detected vehicle. The function can be called as,

```
kalman_def = configureKalmanFilter(MotionModel,
   InitialLocation, ...
           InitialEstimateError, MotionNoise,
              MeasurementNoise)
```

Here, *InitialErrorEstimate* describes the variance of initial estimates of location, velocity and acceleration. The function assumes a zero initial velocity and acceleration for the object at initial position. Increasing the value leads to fast adaptation to the measurements, but also less noise removal. This parameter only affects the first few detections, after which the error is determined by the noise and input data.

*MotionNoise* describes the difference between the object's actual motion and motion model. If this value is increased, the filter might adjust closer to the detection rather than the motion model.

*MeasurementNoise* describes the inaccuracy of the detected location given as input to the filter. This value has to be increased if the measurement is noisy or unreliable. A constant velocity model is chosen for this application.

The *predict* and *correct* functions in Matlab can be used for tracking the object after initializing the Kalman filter. The *predict* function estimates the object's position in the next frame and updates the internal state of the Kalman filter. If the object is detected in the next instant, the state can be corrected or updated by calling the *correct* function with the detected position of the object. This process is repeated at each instant to track the objects, and the object tracking process is summarized in Figure 5.1

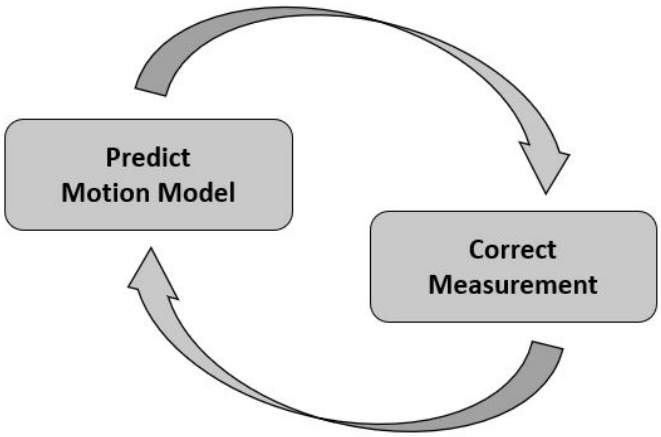



Figure 5.1: Kalman Filter for Object Tracking

### 5.3 Multiple Object Tracking with Kalman Filter

To track multiple objects using a Kalman filter, each object that is detected is assigned an individual track and a Kalman filter. A track can be visualized as a struct object in terms of object-oriented programming, and has fields like ID, Bounding Box, Kalman Filter, Age, Visible and Invisible count. These fields are used to keep track of each object when multiple objects are tracked in a video frame. A track management system must be implemented, which takes care of the following functions:

- Initializing tracks
- Assigning detections to tracks
- Creating new tracks when a new object is detected in the image
- Deleting the Lost Tracks - removing the track(s) that have been invisible for several frames consecutively from the tracking list

### 5.3.1 Assigning Detections to Tracks

Assigning detection to a track is the main step in track management. The first step in assigning a detection is calculating the cost,[1] which can be done using Matlab's *distance* function as,

```
cost = distance(KalmanFilter, centroids)
```

The centroid matrix contains the detected centroids of the vehicles, and each row in the matrix represents a detection. The function calculates the distance between the detected and the predicted position and takes the covariance of the predicted state and process noise into account. A low value for distance implies a low cost of assigning the detection to the track. This function is called after the *predict* function for each track. The calculated costs are arranged in an M x N matrix, where M represents the number of tracks, and N corresponds to the number of detections. The lower the cost, the more likely the detection gets assigned to a track. The cost of not assigning a track or detection must be defined, before assigning detections to tracks. A higher cost means that it is very likely that the algorithm will assign all tracks and detections. This could lead to wrongly assigned tracks and detections. On the other hand, a low cost increases the chances of having new objects being created. The *assignDetectionsToTracks* function uses the Munkres' version of the Hungarian algorithm to compute an assignment which minimizes the total cost. It returns an M x 2 matrix containing the corresponding indices of assigned tracks and detections in its two columns. It also returns the indices of tracks and detections that remained unassigned. The function is called as,

```
[assignments, unassignedTracks, unassignedDetections] = ...
       assignDetectionsToTracks(cost, costOfNonAssignment);
```

[1] Cost is defined as the negative log-likelihood of a detection corresponding to a track.

### 5.4 Snippet - Tracking using Kalman Filter

The following MATLAB code snippet gives an overview of the function calls used in the tracking of vehicles. The entire code for vehicle tracking using Kalman filter, along with the cascade object detector for object detection is presented in the Appendix A for further reading.

```
% Create System objects used for reading video, detecting
   moving
%  objects, and displaying the results.
obj = setupSystemObjects();
tracks = initializeTracks(); % Create an empty array of
   tracks.
nextId = 1; % ID of the next track
% Detect moving objects, and track them across video frames.
while ~isDone(obj.reader)
    frame = readFrame();
    [centroids, bboxes, mask] = detectObjects(frame);
    predictNewLocationsOfTracks();
    [assignments, unassignedTracks, unassignedDetections] =
       ...
        detectionToTrackAssignment();
    updateAssignedTracks();
    updateUnassignedTracks();
    deleteLostTracks();
    createNewTracks();
    displayTrackingResults();
end
```

## 5.5 Results - Kalman Filter based Vehicle Tracking

An image series obtained by tracking multiple objects using the Kalman filter is shown in Figure 5.2. It is seen that the tracking algorithm outputs the vehicle even when it is completely occluded, since it has an estimate of the object's velocity. Apart from this advantage, the number of false positives is also reduced in the final output video as the tracker assigns an object to track only if it is detected continuously for a given number of frames. For a given input video, these parameters can be adjusted to obtain the best possible detection and tracking results.

Figure 5.2: Results of Kalman Filter Vehicle Tracking

# CHAPTER 6
# CONCLUSION AND FUTURE WORK

The concepts of computer vision and machine learning used for vehicle perception in autonomous systems have been illustrated in this thesis along with the implementation details of all the methods explored in object detection and tracking. Some of the notable results obtained from each of these methods have also been shown to give the reader a better understanding of the various approaches used.

## 6.1 Conclusion - Object Detection

The general conclusions from the two object detection methods are:

- The machine learning cascade detector approach explained in Chapter 2 requires a lot of tuning based on the input. A detector trained on a particular set of images cannot be used as an off-the-shelf detector on any input video.
- The DNN based detector performs better than the Cascade Object Detector, and it generalizes well for most inputs.
- For detecting tiny objects, the DNN detector must be trained differently, or should be used in conjunction with the Cascade Object Detector to increase the recall.
- Though the inference time required for the DNN based detector when using a GPU for computation is very small, the time required for processing an image is quite high (~3s/frame). Hence this detector cannot directly be used for real-time vehicle detection without altering the workflow.

## 6.2 Conclusion - Object Tracking

The inferences from the two methods used for tracking vehicles in an image sequence are:

- The optical flow approach explained in Chapter 4 can be used as a technique to estimate the motion of an object in a given image sequence, rather than to track objects in an autonomous vehicle application. The drawbacks of using this method for object tracking are explained in the chapter.
- When tuned well, the Kalman filter can be a good approach for tracking objects. The quality of object tracking depends majorly on the parameters used for setting up the Kalman filter, and so the parameters should be chosen carefully based on the application.

## 6.3 Recommendations for Future Work

The research in this field is fast-paced, and new methods with increased capability are continually being developed, especially based on the publicly available datasets. YOLO V3 is one among the many latest object detection methods that have been developed and released recently. Both YOLO and SSD object detection architectures can be used for real-time object detection in an autonomous driving scenario.

# Appendices

# APPENDIX A

## MATLAB CODE FOR KALMAN TRACKING

```
function Det_Track()
    close all
    clear
    clc
    nextId = 1; % ID of the next track
    %VP = vision.DeployableVideoPlayer;
    VW = vision.VideoFileWriter('Output.avi');
    videoFileLeft = 'GTCampus.avi';
    readerLeft = vision.VideoFileReader(videoFileLeft, '
       VideoOutputDataType', 'uint8');
    tracks = initializeTracks();
    numVehicles = 0;
        %% Loop algorithm
    while ~isDone(readerLeft)
        % Read the frames.
        FrameLeft = readerLeft.step();
        % Detect Cars
        detector = vision.CascadeObjectDetector('cardetector
           .xml');
        [sz,~] = size(FrameLeft);
                %Crop the image to leave out the sky and
                   hood
        tf = 1/4;
```

```
21          bf = 3/4;
22          temparr = FrameLeft(int16(tf*sz:bf*sz),:,:);
23          height = size(temparr, 1) / 3;
24          bboxes = step(detector,temparr);
25          bboxes(:,2) = bboxes(:,2) + 3/2 *height;
26          centroids = [bboxes(:,1)+bboxes(:,3)/2, bboxes(:,2)+
               bboxes(:,4)/2];
27                %Functions for Kalman tracking
28          predictNewLocationsOfTracks()
29          [assignments, unassignedTracks, unassignedDetections
               ] = ...
30          detectionToTrackAssignment();
31          updateAssignedTracks();
32          updateUnassignedTracks();
33          deleteLostTracks();
34          createNewTracks();
35          minVisibleCount = 10;
36          if ~isempty(tracks)
37              % Noisy detections tend to result in short-lived
                    tracks. Only display tracks that have been
                   visible for more than 10 frames.
38              reliableTrackInds = ...
39                  [tracks(:).totalVisibleCount] >
                       minVisibleCount;
40              reliableTracks = tracks(reliableTrackInds);
41              if ~isempty(reliableTracks)
42                  % Get bounding boxes.
```

```
43                  bboxes = cat(1, reliableTracks.bbox);
44                  [badRows, ~] = find(bboxes < 1);
45                  bboxes = int32(bboxes(setdiff(1:size(bboxes
                        ,1),badRows),:));
46              end
47          numVehicles = size(bboxes);
48          numVehicles = numVehicles(1);
49          dispFrame = insertObjectAnnotation(FrameLeft, '
                rectangle', bboxes,...
50              'Vehicle');
51          else
52              dispFrame = FrameLeft;
53          end
54          finalCut = insertText(dispFrame, [10 10], sprintf('
                Vehicles tracked in this frame: %d', numVehicles)
                );
55          %% Update video player
56          %step(VP,finalCut)
57          step(VW,finalCut)
58      end
59  %% Track Initialization
60      function tracks = initializeTracks()
61          % create an empty array of tracks
62          tracks = struct(...
63              'id', {}, ...
64              'bbox', {}, ...
65              'kalmanFilter', {}, ...
```

```
66              'age', {}, ...
67              'totalVisibleCount', {}, ...
68              'consecutiveInvisibleCount', {});
69      end
70
71      function predictNewLocationsOfTracks()
72          for j = 1:length(tracks)
73              bbox = tracks(j).bbox;
74              % Predict the current location of the track.
75              predictedCentroid = predict(tracks(j).
                    kalmanFilter);
76              % Shift the bounding box so that its center is
                    at
77              % the predicted location.
78              predictedCentroid = predictedCentroid - bbox
                    (3:4) / 2;
79              tracks(j).bbox = [predictedCentroid, bbox(3:4)];
80          end
81      end
82
83  %% Assign Detections to Tracks
84  % Assigning object detections in the current frame to
    existing tracks is done by minimizing cost. The cost is
    defined as the negative log-likelihood of a detection
    corresponding to a track.
85      function [assignments, unassignedTracks,
            unassignedDetections] = ...
```

```
86              detectionToTrackAssignment()
87          nTracks = length(tracks);
88          nDetections = size(centroids, 1);
89          % Compute the cost of assigning each detection to
               each track.
90          cost = zeros(nTracks, nDetections);
91          for z = 1:nTracks
92              cost(z, :) = distance(tracks(z).kalmanFilter,
                   centroids);
93          end
94          % Solve the assignment problem.
95          costOfNonAssignment = 50;
96          [assignments, unassignedTracks, unassignedDetections
               ] = ...
97              assignDetectionsToTracks(cost,
                   costOfNonAssignment);
98      end
99
100 %% Update Assigned Tracks
101     function updateAssignedTracks()
102         numAssignedTracks = size(assignments, 1);
103         for i = 1:numAssignedTracks
104             trackIdx = assignments(i, 1);
105             detectionIdx = assignments(i, 2);
106             centroid = centroids(detectionIdx, :);
107             bbox = bboxes(detectionIdx, :);
108             % Correct the estimate of the object's location
```

```
109             % using the new detection.
110             correct(tracks(trackIdx).kalmanFilter, centroid)
                    ;
111             % Replace predicted bounding box with detected
112             % bounding box.
113             tracks(trackIdx).bbox = bbox;
114             % Update track's age.
115             tracks(trackIdx).age = tracks(trackIdx).age + 1;
116             % Update visibility.
117             tracks(trackIdx).totalVisibleCount = ...
118                 tracks(trackIdx).totalVisibleCount + 1;
119             tracks(trackIdx).consecutiveInvisibleCount = 0;
120         end
121     end
122
123 %% Update Unassigned Tracks
124     function updateUnassignedTracks()
125         for p = 1:length(unassignedTracks)
126             ind = unassignedTracks(p);
127             tracks(ind).age = tracks(ind).age + 1;
128             tracks(ind).consecutiveInvisibleCount = ...
129                 tracks(ind).consecutiveInvisibleCount + 1;
130         end
131     end
132
133 %% Delete Lost Tracks
134     function deleteLostTracks()
```

```
135         if isempty(tracks)
136             return;
137         end
138         invisibleForTooLong = 10;
139         ageThreshold = 8;
140         % Compute the fraction of the track's age for which
               it was visible.
141         ages = [tracks(:).age];
142         totalVisibleCounts = [tracks(:).totalVisibleCount];
143         visibility = totalVisibleCounts ./ ages;
144         % Find the indices of 'lost' tracks.
145         lostInds = (ages < ageThreshold & visibility < 0.6)
               | ...
146             [tracks(:).consecutiveInvisibleCount] >=
                   invisibleForTooLong;
147         % Delete lost tracks.
148         tracks = tracks(~lostInds);
149     end
150
151 %% Create New Tracks
152 % Create new tracks from unassigned detections. Assume that
       any unassigned detection is a start of a new track.
153     function createNewTracks()
154         centroids = centroids(unassignedDetections, :);
155         bboxes = bboxes(unassignedDetections, :);
156         for i = 1:size(centroids, 1)
157             centroid = centroids(i,:);
```

```
158             bbox = bboxes(i, :);
159             % Create a Kalman filter object.
160             kalmanFilter = configureKalmanFilter('
                   ConstantVelocity', ...
161                 centroid, [200, 50], [100, 25], 100);
162             % Create a new track.
163             newTrack = struct(...
164                 'id', nextId, ...
165                 'bbox', bbox, ...
166                 'kalmanFilter', kalmanFilter, ...
167                 'age', 1, ...
168                 'totalVisibleCount', 1, ...
169                 'consecutiveInvisibleCount', 0);
170             % Add it to the array of tracks.
171             tracks(end + 1) = newTrack;
172             % Increment the next id.
173             nextId = nextId + 1;
174         end
175     end
176     %% Clean up
177     reset(readerLeft);
178     %release(VP)
179     release(VW)
180 end
```

## APPENDIX B

## CODE FOR DNN BASED VEHICLE DETECTION

```
1  using namespace std;
2  using namespace dlib;
3  const string dir = "~/thesis/png_in/";
4  const string out_dir = "~/thesis/png_out/";
5  // The rear view vehicle detector network
6  template <long num_filters, typename SUBNET> using con5d =
       con<num_filters,5,5,2,2,SUBNET>;
7  template <long num_filters, typename SUBNET> using con5  =
       con<num_filters,5,5,1,1,SUBNET>;
8  template <typename SUBNET> using downsampler  = relu<affine<
       con5d<32, relu<affine<con5d<32, relu<affine<con5d<16,
       SUBNET>>>>>>>>>;
9  template <typename SUBNET> using rcon5  = relu<affine<con5
       <55,SUBNET>>>;
10 using net_type = loss_mmod<con<1,9,9,1,1,rcon5<rcon5<rcon5<
       downsampler<input_rgb_image_pyramid<pyramid_down
       <6>>>>>>>>;
11
12 int main() try
13 {
14     net_type net;
15     shape_predictor sp;
```

```
16      // You can get this file from http://dlib.net/files/
            mmod_rear_end_vehicle_detector.dat.bz2
17      // This network was produced by the
            dnn_mmod_train_find_cars_ex.cpp example program.
18      // As you can see, it also includes a shape_predictor.
            To see a generic example of how
19      // to train those refer to train_shape_predictor_ex.cpp.
20      deserialize("mmod_rear_end_vehicle_detector.dat") >> net
            >> sp;
21          int n = 2002;
22      matrix<rgb_pixel> img;
23          cv::Mat out_img(1,512,CV_32F);
24
25          for (int i = 1; i < n; i++){
26                  std::string imgseq = dir +"frame"+to_string(
                        i)+".png";
27                  load_image(img, imgseq);
28                  std::string imgseq_out = out_dir +"frameout"
                        +to_string(i)+".png";
29
30      // Run the detector on the image and show us the output.
31          for (auto&& d : net(img))
32          {
33              auto fd = sp(img,d);
34              rectangle rect;
35              for (unsigned long j = 0; j < fd.num_parts(); ++
                    j)
```

```
36                  rect += fd.part(j);
37              draw_rectangle(img,rect,rgb_pixel(255,0,0),1);
38          }
39                  out_img = dlib::toMat(img);
40          switch(cv::waitKey(1))
41          {
42              case 27:
43                  return 0;
44          }
45                  cv::cvtColor(out_img,out_img,cv::
                        COLOR_RGB2BGR);
46                  cv::imwrite(imgseq_out,out_img);
47      }
48  }
49  catch(image_load_error& e)
50  {
51      cout << e.what() << endl;
52      cout << "The test image is located in the examples
            folder.  So you should run this program from a sub
            folder so that the relative path is correct." << endl
            ;
53  }
54  catch(serialization_error& e)
55  {
56      cout << e.what() << endl;
57      cout << "The model file can be obtained from: http://
            dlib.net/files/mmod_rear_end_vehicle_detector.dat.bz2
```

```
            Don't forget to unzip the file." << endl;
58  }
59  catch(std::exception& e)
60  {
61      cout << e.what() << endl;
62  }
```

## REFERENCES


[1] P. Viola and M. Jones, "Rapid object detection using a boosted cascade of simple features," *Proc. IEEE Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. I–511–I–518, 2001.

[2] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," *Proc. IEEE Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 1, pp. 886–893, 2005.

[3] S. Ren, K. He, R. Girshick, and J. Sun, "Faster r-cnn: Towards real-time object detection with region proposal networks," *Advances in neural information processing systems*, pp. 91–99, 2015.

[4] W. Liu, D. Anguelov, D. Erhan, C. Szegedy, S. Reed, C.-Y. Fu, and A. C. Berg, "SSD: Single shot multibox detector," *European Conference on Computer Vision*, pp. 21–37, 2016.

[5] MathWorks, *Train a cascade object detector*, `https://www.mathworks.com/help/vision/ug/train-a-cascade-object-detector.html`.

[6] OpenCV, *Opencv api reference*, `https://docs.opencv.org/2.4/modules/objdetect/doc/objdetect.html`.

[7] P. Sermanet, D. Eigen, X. Zhang, M. Mathieu, R. Fergus, and Y. LeCun, "Overfeat: Integrated recognition, localization and detection using convolutional networks," *International Conference on Learning Representations (ICLR)*, 2014.

[8] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proc. IEEE Soc. Conf. Comput. Vis. Pattern Recognit.(CVPR)*, 2014.

[9] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Soc. Conf. Comput. Vis. Pattern Recognit.(CVPR)*, 2016.

[10] B. Horn and B. Schunck, "Determining optical flow," *Artificial Intelligence*, pp. 185–204, 1981.

[11] MathWorks, *Motion estimation*, `https://www.mathworks.com/help/vision/motion-estimation.html`.

[12] ——, *Multiple object tracking*, https://www.mathworks.com/help/vision/ug/multiple-object-tracking.html.